НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра автоматизованих систем обробки інформації і управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)

"___"_____2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «*Система управління агентами у відео грі*»

Виконав: студент *IV* курсу, групи *ІС-з61*

Фокін Андрій Ігорович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доцент, к.т.н, Коган Алла Вікторівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з норм контролю *доцент, к.т.н, доцент Тєлишева Тамара Олексіївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *доцент, к.т.н, Роковий Олександр Петрович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)

“___”_____2020 р.

З А В Д А Н Н Я
на дипломний проєкт студенту

Фокін Андрій Ігорович
(прізвище, ім’я, по батькові)

1. Тема проєкту «Система управління агентами у відео грі»

керівник проєкту *Коган Алла Вікторівна, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*7*”*травня* 2020 р. №1081-с
2. Термін подання студентом проєкту “*08*”*червня* 2020 року
3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання
4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів
5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна діяльності

2. Схема структурна варіантів використання

3. Креслення вигляду екранних форм

4. Схема бази даних

5. Схема структурна частини АС

6. Схема структурна послідовності

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *«13» квітня 2020 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*14.04.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*15.04.2020*	
3.	*Постановка та формалізація задачі*	*17.04.2020*	
4.	*Розробка інформаційного забезпечення*	*20.04.2020*	
5.	*Алгоритмізація задачі*	*22.04.2020*	
6.	*Обґрунтування використовуваних технічних засобів*	*22.04.2020*	
7.	*Розробка програмного забезпечення*	*30.04.2020*	
8.	*Налагодження програми*	*07.05.2020*	
9.	*Виконання графічних документів*	*10.05.2020*	
10.	*Оформлення пояснювальної записки*	*12.05.2020*	
11.	*Подання ДП на попередній захист*	*15.05.2020*	
12.	*Подання ДП на основний захист*	*01.06.2020*	
13.	*Подання ДП рецензенту*	*02.06.2020*	

Студент Андрій ФОКІН

Керівник Алла КОГАН

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6110.00.000 ПЗ	Пояснювальна записка	51	
3	А4	ДП 6110.01.000 ТЗ	Технічне завдання	9	
4	А3	ДП 6110.02.000 СО	Схема організаційної структури	1	
5	А3	ДП 6110.03.000 ССВ	Схема структурна варіантів використання	1	
6	А3	ДП 6110.04.000 КЕ	Креслення вигляду екранних форм	1	
8	А3	ДП 6110.06.000 СК	Схема компонентів	1	
9	А3	ДП 6110.07.000 ССП	Схема структурна послідовності	1	
10	А3	ДП 6110.08.000 ССК	Схема структурна класів програмного забезпечення	1	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП з6110 00.000.00		
Розробн.		Фокін А. І.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Коган А.В.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-з61	
Н/контр.		Телешева Т.О					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: Система управління агентами у відео грі

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з шести розділів, містить 18 рисунків, 1 таблиця, 1 додатків, 11 джерел.

Дипломний проєкт присвячений створенню системи управління агентами у відео грі.

Метою створення системи є спрощення процесу створення відео ігор за рахунок створення системи управління агентами у відео грі.

У розділі загальних положень описано процес діяльності штучного інтелекту, складено функціональну модель системи, описано відмінності від існуючих аналогів. Сформовано мету розробки та визначено задачі, які необхідно вирішити.

У розділі інформаційного забезпечення описано вхідні та вихідні дані.

Розділ математичного забезпечення присвячений обґрунтуванню обраних методів розв'язання задачі, а також опису алгоритмів на базі яких розроблятиметься система.

У розділі програмного забезпечення описуються засоби розробки програмного продукту та етапи проектування архітектури. Описано специфікацію функцій та звіти, що генеруються в ході роботи програми.

Технологічний розділ визначає мету проведення випробувань програмного продукту та описує їх результати.

ШТУЧНИЙ ІНТЕЛЕКТ, ВІДЕО ГРА, АГЕНТ, ПОШУК ШЛЯХУ, A*, ГРАВЕЦЬ

					ДП 6110.00.000 ПЗ		
Зм.	Арк.	Прізвище	Підпис	Дата			
Розроб.		Фокін А.І			Система управління агентами у відео грі	Літ.	Лист 2 / Листів
Перевірив.		Коган А.В					
Н. кон.		Тєлишева Т.О.				КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-з61	
Затв.		Павлов О.А.					

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of six sections, contains 18 figures, 1 table, 1 appendices, 11 sources.

The diploma project is devoted to the creation of an agent management system in a video game.

The purpose of creating a system is to simplify the process of creating video games by creating an agent management system in a video game.

In the section of general provisions the process of activity of artificial intelligence is described, the functional model of system is made, differences from existing analogues are described. The purpose of development is formed and the tasks which need to be solved are defined.

The information support section describes the input and output data.

The section of mathematical software is devoted to the substantiation of the chosen methods of solving the problem, as well as the description of the algorithms on the basis of which the system will be developed.

The software section describes the software development tools and architecture design steps. Describes the specification of functions and reports generated during the program.

The technology section defines the purpose of testing the software product and describes their results.

ARTIFICIAL INTELLIGENCE, VIDEO GAME, AGENT, PATHFINDING, A *, PLAYER

ЗМІСТ

ВСТУП

Зі створенням першого програмованого комп'ютера відразу після закінчення Другої Світової Війни (1940-1950 рр.) Виникла область науки вивчає комп'ютерні програми, що мають здатність "мислити" в тому чи іншому сенсі. Цю науку і технологію назвали Artificial intelligence. Її основоположниками є багато вчених, які стоять за створенням тих же перших комп'ютерів (Алан Тюрінг, Джон Маккарті, Марвін Мінськ та ін.)[1].

Історія AI знала більше 5 періодів занепаду (т.зв. "Зима в AI") і стільки ж відроджень, щоразу змінюючи вектор в вивчення AI. По-цьому до цих пір ця наука є областю в якій є безліч питань без відповідей, а домогтися результату може кожен (Russell and Norvig, 1995), на відміну від старіших і фундаментальних областей на зразок фізики та хімії. "Штучний інтелект, з іншого боку, все ще відкриває можливості для прояву талантів декількох справжніх Ейнштейнов "(Russell and Norvig, 1995) Це все робить AI для автора виключно цікавою областю для вивчення.

Практично відразу після заснування AI з'явилися перші програми, які грають у гри. У 1957-му році Крістофер Стречі написав гру, що грає в шашки. Тоді ж був написаний перший AI, примітивно грає в шахи. (Russell and Norvig, 1995) ШІ в комп'ютерних іграх з тих пір значно розвинувся. Проте, шахову програму сильніше людини змогли написати лише в 1997-му році, коли DeepBlue здобула перемогу над Гаррі Каспаровим. Це стало можливим лише завдяки збільшенню комп'ютерних потужностей. Але цей успіх був можливий також завдяки новим алгоритмам і підходам. У будь-якому випадку не варто думати, що розвиток комп'ютерної техніки є "срібною кулею" у вирішенні всіх проблем в комп'ютерних AI. Наприклад, гра "Го" до сих пір є невирішеною завданням для програмістів AI. Найрозумніша на сьогоднішній день грає в неї на аматорському рівні. (Russell and Norvig, 1995)

Сьогодні індустрія комп'ютерних ігор практично не обходиться без застосування AI. Він в іграх, на ряду з графічними та іншими характеристиками, є важливим якісним показником і безпосередньо впливає на їх конкурентно-спроможність і успіх. Це робить ігровий AI затребуваним. Ігри за своєю природою найчастіше мають не тільки розвиває, але й розважальну складову, по-цьому автору дана тема особливо цікава.

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Опис предметного середовища

Опис процесу діяльності

Якщо б ви запитали шанувальників відеоігор, як може виглядати ідеалізований, ще не можливий фрагмент інтерактивної розваги через 10, а то й 20 років, вони могли б описати щось моторошно, подібне до програмного забезпечення, яке міститься в науково-фантастичній класиці Орсона Скотта Карда під назвою Гра Ендера. У своєму романі Карт уявив симуляцію військового класу, якою керує передовий, непереборний штучний інтелект.

The Mind Game призначена насамперед для вимірювання психологічного стану молодих новобранців, і часто представляє своїм гравцям неможливі ситуації для перевірки їх душевної стійкості перед неминучим ураженням. Однак гра також є нескінченною процедурою, створюючи навколишнє середовище та ситуації на ходу, і дозволяє гравцям виконувати будь-які дії у віртуальному світі, які могли б могли статись у реальному. Ідучи ще далі, він реагує на емоційний та психологічний стан своїх гравців, адаптуючись та реагуючи на поведінку людини та розвиваючись з часом. Одного разу гра «Розум» навіть звертається до спогадів гравця, щоб генерувати цілі світи гри, пристосовані до минулого Ендера.

The Mind Game - це вагома відправна точка для розмови про майбутнє відеоігор та штучного інтелекту. Чому гри і AI використовуються як для їх створення, так і для управління діями віртуальних персонажів, навіть якщо вони не такі складні? І які інструменти або технології ще потрібні розробникам для досягнення цього гіпотетичного злиття II і симульованої реальності?

Це питання, які дослідники та дизайнери ігор тільки зараз починають вирішувати, коли останні досягнення в галузі AI починають переходити від експериментальних лабораторій і перетворюватися на комерційні продукти та корисні інструменти розробки. До цього часу AI здатний до самонавчання – а саме глибокий навчальний набір революції машинного навчання – це призвело до прогресу в автоматичному керуванні автомобілями, комп'ютерного зору та обробки природніх мов, насправді не переросло в розвиток комерційних ігор. Це незважаючи на те, що деякі з цих успіхів в AI частково дякують програмному забезпеченню, яке вдосконалювалося завдяки дії відеоігор, наприклад, неперевершеній програмі AlphaGo DeepMind та боту Dota 2 OpenAI, який тепер може перемагати гравців гравців професійного рівня.

Опис функціональної моделі

У цьому модулі розглянуто розумову модель високого рівня різних розділів AI відеоігри.

Щоб штучний інтелект міг приймати продумані рішення, йому необхідно якимось чином сприймати середу, в якій він знаходиться. У простих системах таке сприйняття може бути банальною перевіркою положення агента гравця з ігровому світі. У складніших системах необхідно визначати головні характеристики та властивості світу гри, наприклад вірогідні шляхи для пересування агентів, наявність статичних та динамічних укриттів на ігрових рівнях, області конфліктів з агентами.

При цьому розробники повинні придумувати шляхи виявлення і визначення головних властивостей світу гри, важливих для системи штучного інтелекту. Наприклад, укриття на місцевості можуть бути визначені наперед дизайнерами рівнів або заздалегідь оброблені при завантаженні або компіляції ігрової локації. Деякі елементи потрібно

обчислювати на льоту, наприклад карти конфліктів та найбільш вірогідні або найближчі загрози.

Огляд наявних аналогів

Протягом останніх декількох років ігрова індустрія зазнала низки змін, і ігри відіграли ключову роль у дослідженні AI, поставивши цікаві проблеми, які потрібно вирішити. Але це не лише прогресування AI завдяки іграм, а інтеграція AI також отримала користь від ігрової індустрії. Фактично, AI покращив ігри на декількох фронтах - від дизайну ігор, розробки до функціональності і навіть того, як в нього грають. Програмне забезпечення AlphaGo DeepMind, більш відоме тим, що перемогло дійсного чемпіона Go.

Ігровий AI, який нагадує AI, вже давно є в просторі ігрових розробок. Аспект AI в грі унікальний у кожній з них. AI програмується по-різному з точки зору ігор. Саме від машин Кінцевого стану в Pac-man до алгоритму A * в Маріо, внесок ігрового AI в покращення гри був величезним.

Далі в розділі перераховано найкращі відеоігри, які мають найкращий AI, включений у свій ігровий процес. Відеоігри подаються не в конкретному порядку.

Tom Clancy's Splinter Cell: Blacklist. Геймери називають серію Splinter Cell революцією. Splinter Cell - це екшн-пригодницька гра, яка спирається на механіку непомітності та стратегію в ігровому процесі. Гравець грає як Сем Фішер, висококваліфікований агент підрозділу розвідки. Типовий шутер від третьої особи, гравець планує свої кроки для того, щоб виконувати кожен рівень гри.

Нещодавнє додаток, Splinter Cell: Blacklist рекламується як найкраща частина серії. AI реагує на кожен зроблений гравцем хід і діє відповідно. Нехай це будуть різні маневри в грі, але "Guard AI" - просто виняткове.

Rocket League - це футбольна відеоігра, в якій представлені автомобілі з ракетами. Випущена в 2015 році, вона все ще є однією з найбільш

популярних багатокористувацьких онлайн ігор. У грі гравці вбивають м'яч у ворота суперника автомобілями з ракетними двигунами. Хоча це схоже на будь-яку спортивну відео-гру, гра AI дуже потужна для управління стратегією гри з м'ячем. Це забезпечує реалістичний досвід. Ідея поєднання суперництва із захопленням футболом та автомобілями дуже добре вийшла.

First Encounter Assault Recon, більш відомий як F.E.A.R, - шутера від першої особи (FPS), розроблена Monolith Productions. Випущена в 2005 році, ця гра привернула багато уваги в ігровому співтоваристві за свій ігровий AI. Насправді він може позмагатися навіть з сучасними AI у цьому жанрі.

Приступаючи до геймплея, це типовий тактичний шутер з моторошним сюжетом. Як вже згадувалося раніше, це відрізняє противника AI, який надзвичайно хороший і реагує навіть на дрібні деталі. Наприклад, у грі, якщо гравець прикривається барикадою, ворожий AI швидко усвідомлює це і кидає гранату для усунення гравця. Ще один аспект - це спілкування AI у грі. Вороги розмовляють та кординують свої рухи відповідно до руху гравців.

F.E.A.R досягнув того, чого AI не міг досягти в попередні роки, а саме досвід, схожий на гру проти людини. Навіть сьогодні гра є одним з найбільших шутерів Game AI.

The Last of Us – це пригодницька відео-гра, опублікована компанією Sony Interactive Entertainment у 2013 році. У грі висвітлюється історія дуету вцілілих людей, які подорожують, щоб знайти ліки від чуми, яка заразила більшість людей у США. Ігровий процес подається з виглядом від третьої особи, оскільки гравець проходить безліч завдань і сценаріїв, розповсюджених по ігровому світу.

Приступаючи до ігрового AI, кожен персонаж має два набори поведінкових рис – для сторони що атакує та на стороні оборони. Все залежить від того, як гравці реагують на ситуацію, в якій пов'язані різні персонажі. AI перемикається на основі цих відповідей і діє відповідно. Однак

"Останній з нас" отримував критику, коли AI веде себе дивно. Тим не менш, ігрові патчі виправили ці проблеми.

Постановка задачі

Призначення розробки

Система призначена для управління агентами у відео грі.

Цілі та задачі розробки

Метою розробки є спрощення процесу управління агентами у відео грі за рахунок автоматизації створення моделей поведінки агентів із застосуванням методів передбачення поведінки гравця.

Завдяки цьому задіяні агенти можуть гнучко взаємодіяти з середовищем гри і самим гравцем чи декількома гравцями в залежності від потреб гри, для якої застосовується система, чи окремої ігрової ситуації, в якій агенти приймають участь.

Для досягнення мети необхідно вирішити такі задачі:

- збір інформацій про середовище в якому знаходиться агент;
- обробка інформації та прийняття рішення на її основі;
- взаємодія з ігровим світом чи гравцем на основі прийнятого рішення.

Висновок до розділу

В процесі написання даного розділу було описано предметне середовище. Встановлено, що ігрова індустрія має великий потенціал для досліджень, а також складова штучного інтелекту та управління ігровими агентами, в цій індустрії, створює багато задач для автоматизації.

Також було описано процес діяльності, який автоматизується, визначено, які етапи потрібно пройти, для розв’язування задачі.

Було описано функціональну модель системи і визначено функції системи, а також проведено аналіз наявних аналогів та встановлено. Сформульовано призначення розробки, визначено мету та визначено задачі, які необхідно роз'вязати для досягнення мети.

ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

Вхідні дані

Як вхідні данні у системі управління агентами використовуються інформація про геометрію ігрової локації в якій знаходиться агент. Такі данні можливо поділити на дві категорії:

- статичні;
- динамічні.

Статичні ігрові данні задаються на етапі конструювання гри як форми геометрії в гра та можливі типи взаємодії з цими формами. Наприклад NavMesh.

Динамічні це данні які вираховуються під час самої гри і є непередбачуваними до початку гри. Наприклад як буде переміщуватись гравець.

Вихідні дані

Система не містить вихідних даних у явній формі. Отриманим результатом слугує результуюча поведінка агентів у грі та логіка їхньої взаємодії з гравцем.

Структура масивів інформації

Ігровий процес, керований даними, допомагає зменшити обсяг роботи та складності, а також надає можливість візуалізувати та параметризувати створення та просування даних для ігор, які мають триваліший термін експлуатації набагато більше, ніж типова гра в бокс, і вимагають постійного налаштування та врівноваження даних на основі відгуків гравців.

Можливість переміщення даних у Microsoft Excel або інші документи з електронних таблиць, які можна підтримувати за допомогою перевірених інструментів, а потім імпортувати для автоматичного введення в дію гри.

Є два нові способи імпорту даних у UE4 через документи Excel:

- curveTables;
- dataTables.

DataTables, як випливає з назви, - це таблиця різних, але пов'язаних даних, згрупованих за змістом і корисним способом, де полями даних можуть бути будь-які дійсні властивості UObject, включаючи посилання активів. Перш ніж дизайнер зможе імпортувати файл CSV в таблицю даних, програміст повинен створити контейнер рядків, який повідомляє двигуну, як інтерпретувати дані. Ці таблиці складаються з імен стовпців, які мають зіставлення 1: 1 із заданим кодом на основі UStruct та його змінних, які повинні успадковуватися від FTableRowBase[12].

	0	1	2	3
Melee_Damage	15	20	25	30
Melee_KnockBack	1	2	4	8
Melee_KnockBackAngle	10	45	60	65
Melee_StunTime	0	1	5	7

Рисунок 2.1 – Приклад DataTable.

DataCurves працюють подібним чином до DataTables, проте вони підтримують лише дробні числа.

Висновок до розділу

У процесі написання цього розділу було визначено вхідні та вихідні дані системи, що створюється.

Надано вичерпний опис вхідних даних, що не задаються користувачем, а визначаються програмно, в залежності від конкретного алгоритму для створення моделей.

МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

Змістовна постановка задачі

Алгоритми пошуку шляху з підручників з інформатики працюють з графами в математичному сенсі - набором вершин з ребрами, що з'єднують їх. Плиточная ігрова карта може розглядатися як граф, де кожна плитка являє собою вершину, а ребра намальовані між плитками, які знаходяться поруч один з одним (рис. 3.1):

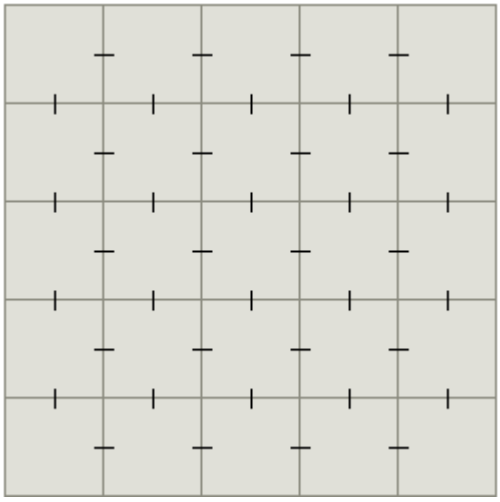


Рисунок 3.1 – Середовище проведення конкурсу.

Більшість алгоритмів пошуку шляху з досліджень AI або Algorithms призначені для довільних графів, а не для ігор на основі сітки. Ми хотіли б знайти щось, що могло б скористатися природою ігрової карти. Є деякі речі, які ми вважаємо здоровим глуздом, але які алгоритми не розуміють. Ми знаємо дещо про відстані: в загальному, у міру того, як дві речі розходяться один від одного, перехід від одного до іншого займе більше часу, за умови, що немає червоточини. Ми знаємо дещо про напрямки: якщо ви прямуєте на схід, то краще за все знайти шлях, якщо йти пішки на схід, ніж йти на захід. У сітках ми знаємо дещо про симетрії: більшу частину часу рух на північ, а потім на схід, рівнозначно руху на схід, а потім на північ. Ця додаткова

інформація може допомогти нам прискорити виконання алгоритмів пошуку шляхів.

Математична постановка задачі

Нехай маємо множину вершин *D*, що складається з умовних точок можливого шляху агента, де кожній точці відповідає позиція у дво- або трьох-вимірному ігровому просторі :

$$D = \{d_1, d_2, .. d_n\}, \quad (3.1)$$

де d_i – конкретна вершина, $i = \overline{1,n}$, n – кількість вершин.

Нехай маємо множину переходів C, які з'єднують вершини з множини D:

$$C = \{c_1, c_2, .. c_m\}, \quad (3.2)$$

де $c_i, i = \overline{1,m}$ конкретний перехід, m– кількість переходів.

Множини D і C задають навігаційний граф G по якому переміщуються агенти. Також таку структуру називають навігаційною сіткою (NavMesh).

Задача пошуку шляху полягає у знаходженні послідовності з вершин, де першим елементом є початок шляху а останнім шукана позиція.

Опис методів розв'язання

Алгоритм Дейкстри і Best-First-Search

Алгоритм Дейкстри працює, відвідуючи вершини графа, починаючи з початкової точки об'єкта. Потім він багаторазово досліджує найближчу, ще не досліджену вершину, додаючи її вершини до набору вершин, які потрібно досліджувати. Він розширюється назовні від початкової точки, поки не досягне мети. Алгоритм Дейкстри гарантовано знайде найкоротший шлях від початкової точки до мети, якщо жодне з ребер не має негативної вартості. (Я пишу «найкоротший шлях», тому що часто є кілька еквівалентно коротких шляхів.) На наступній діаграмі рожевий квадрат - це відправна точка, синій

квадрат - це мета, а бірюзові області показують, які області сканував алгоритм Дейкстри. Найсвітліші області бірюзового кольору є найбільш далекими від початкової точки і, таким чином, утворюють «кордон» дослідження (рис. 3.2):

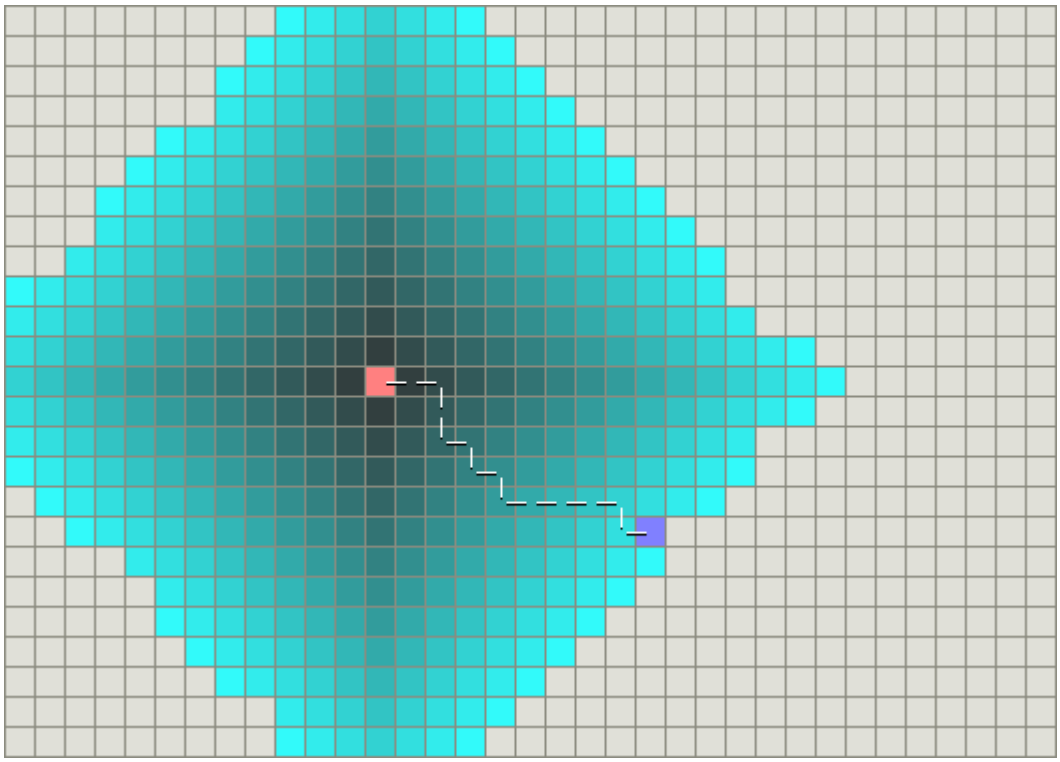


Рисунок 2.2 – Приклад роботи алгоритму Дейкстри.

Алгоритм Greedy Best-First-Search працює аналогічним чином, за винятком того, що він має деяку оцінку (звану евристикою) того, наскільки далеко від цілі знаходиться будь-яка вершина. Замість вибору вершини, найближчій до початкової точки, він вибирає вершину, найближчу до мети. Greedy Best-First-Search не гарантує знаходження найкоротшого шляху. Тим не менш, він працює набагато швидше, ніж алгоритм Дейкстри, тому що він використовує евристичну функцію, щоб дуже швидко направити свій шлях до мети. Наприклад, якщо мета знаходиться на південь від початкової позиції, Greedy Best-First-Search буде прагнути зосередитися на шляхах, що ведуть на південь. На наступній діаграмі жовтий являє вузли з високим евристичним значенням (висока вартість для досягнення мети), а чорні

представляють вузли з низьким евристичним значенням (низька вартість для досягнення мети). Це показує, що Greedy Best-First-Search може знайти шляхи дуже швидко в порівнянні з алгоритмом Дейкстри (рис. 3.3):

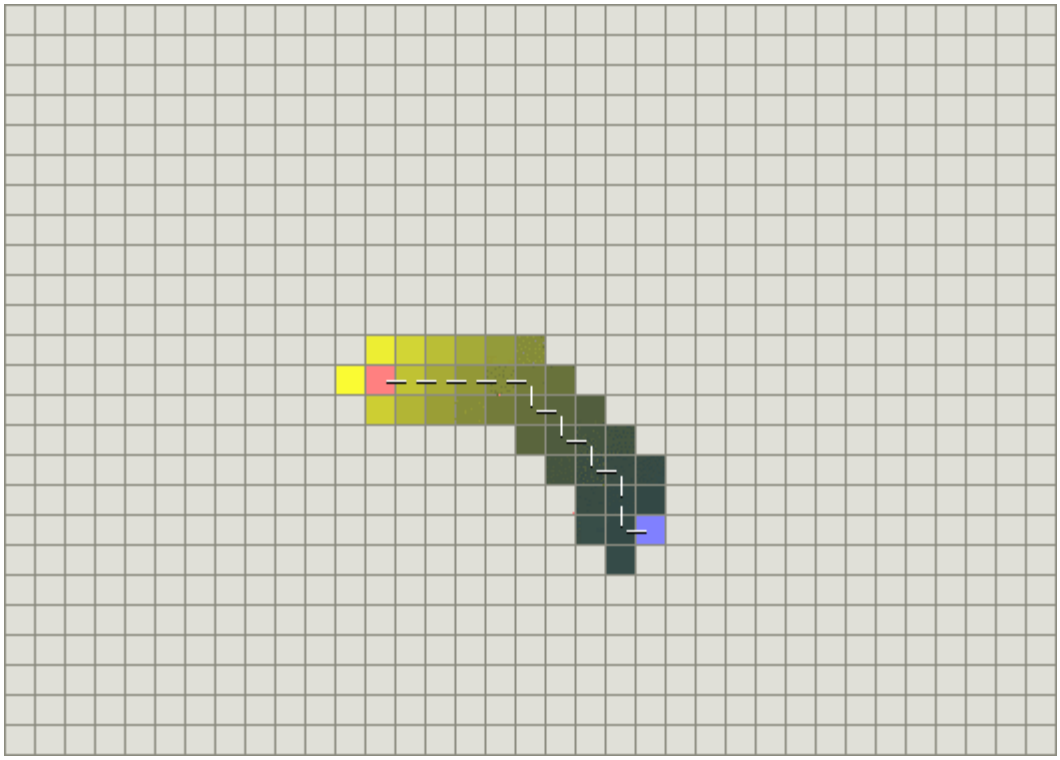


Рисунок 3.3 – Приклад роботи алгоритму Greedy Best-First-Search.

Однак обидва ці приклади ілюструють найпростіший випадок -, коли на карті немає перешкод, а найкоротший шлях дійсно є прямою лінією. Давайте розглянемо увігнуте перешкоду, як описано в попередньому розділі. Алгоритм Дейкстри працює старанніше, але гарантовано знайде найкоротший шлях (рис. 3.4):

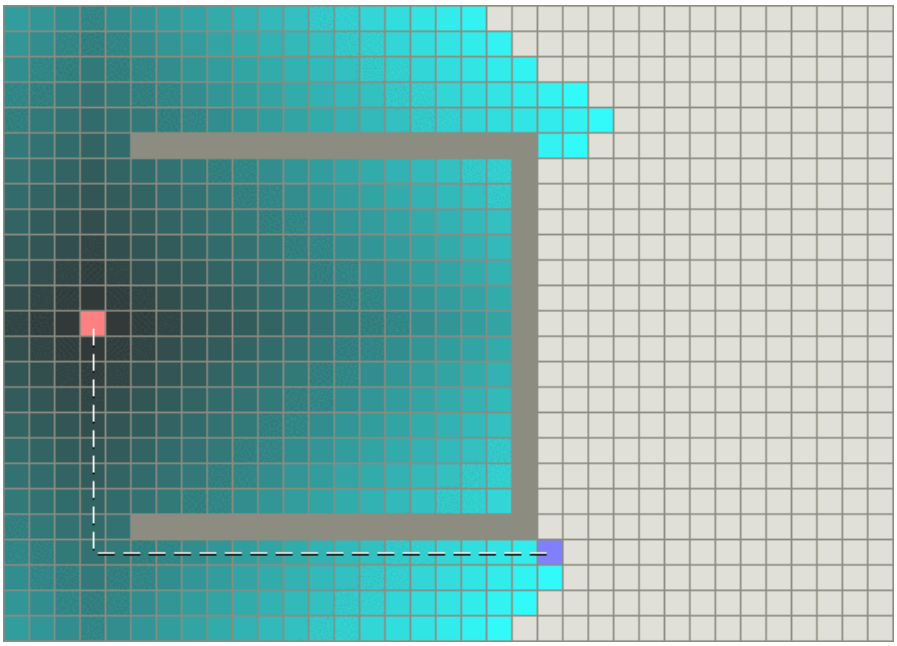


Рисунок 3.4 – Приклад роботи алгоритму Дейкстри з перешкодами.

З іншого боку, Greedy Best-First-Search виконує менше роботи, але його шлях явно не такий гарний (рис. 3.5):

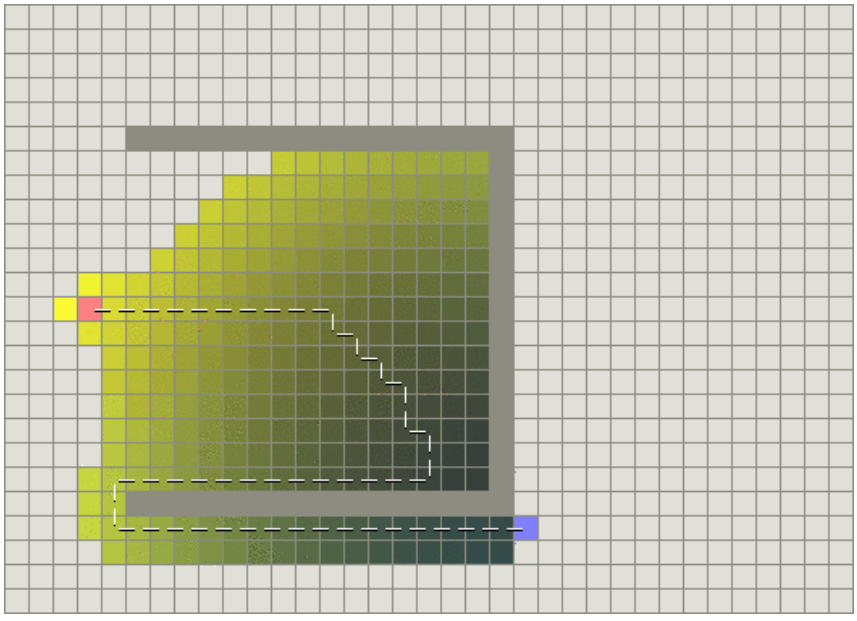


Рисунок 3.5 – Приклад роботи алгоритму Greedy Best-First-Search з перешкодами.

Проблема в тому, що Greedy Best-First-Search є «жадібним» і намагається рухатися до мети, навіть якщо це неправильний шлях. Оскільки

він враховує тільки вартість досягнення мети і ігнорує вартість шляху, він продовжує йти, навіть якщо шлях, по якому він йде, став дійсно довгим.

Хіба не було б непогано об'єднати найкраще з обох? A * був розроблений в 1968 році для об'єднання евристичних підходів, таких як Greedy Best-First-Search, і формальних підходів, таких як алгоритм Dijsktra. Це трохи незвично в тому сенсі, що евристичні підходи зазвичай дають вам приблизний спосіб вирішення проблем, не гарантуючи, що ви отримаєте найкращий відповідь. Однак A * побудований поверх евристики, і хоча сама евристика не дає вам гарантії, A * може гарантувати найкоротший шлях.

Алгоритм A*

Я зосереджуся на алгоритмі A*. A* – найпопулярніший вибір для пошуку шляху, тому що він досить гнучкий і може використовуватися в широкому діапазоні контекстів[13].

A* подібний алгоритму Дейкстри в тому сенсі, що його можна використовувати для пошуку найкоротшого шляху. A* схожий на Greedy Best-First-Search в тому сенсі, що він може використовувати евристику для управління собою. У простому випадку це так само швидко, як Greedy Best-First-Search (рис. 3.6):

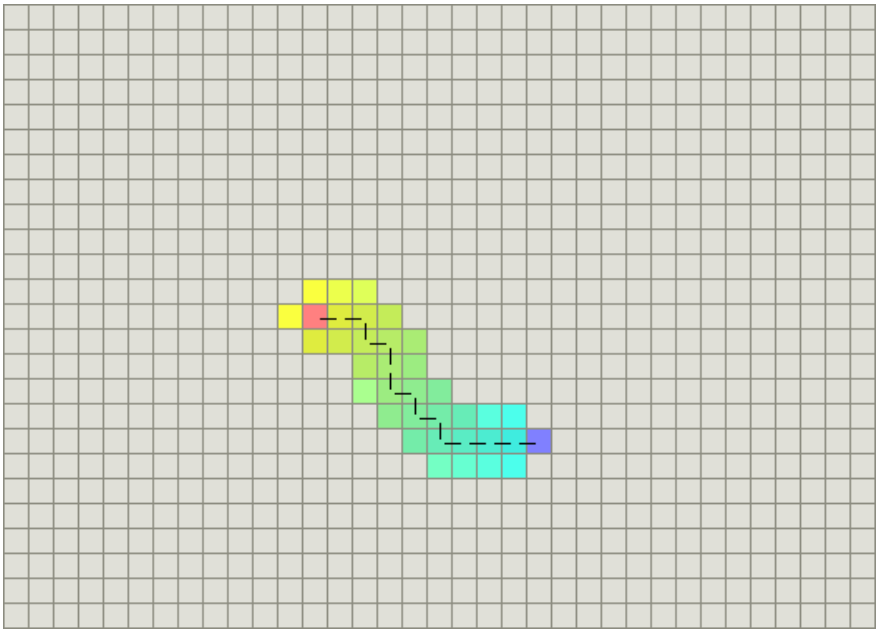


Рисунок 3.6 – Приклад роботи алгоритму A*.

У прикладі з увігнутим перешкодою A * знаходить шлях так само добре, як це знайшов алгоритм Дейкстри (рис. 3.7):

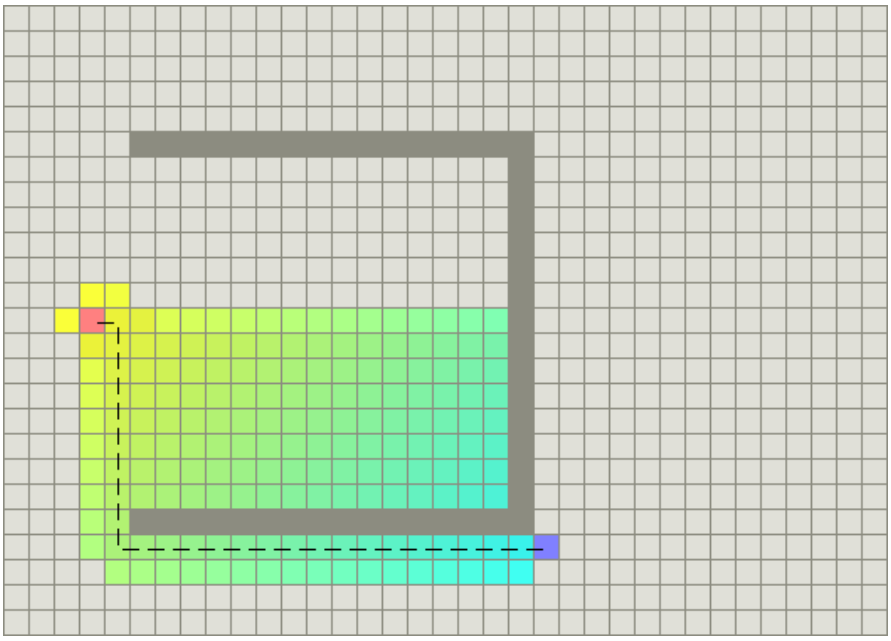


Рисунок 3.7 – Приклад роботи алгоритму A* з перешкодами.

Секрет його успіху в тому, що він об'єднує інформацію, яку використовує алгоритм Дейкстри (вважаючи за краще вершини, близькі до початкової точки), і інформацію, яку використовує Greedy Best-First-Search (вважаючи за краще вершини, близькі до мети). У стандартній термінології,

використовуваної при розмові про A *, g (n) представляє точну вартість шляху від початкової точки до будь-якої вершини n, a h (n) представляє евристичну оціночну вартість від вершини n до мети. На наведених вище діаграмах жовтий (h) являє вершини, віддалені від мети, а чирок (g) представляє вершини, віддалені від початкової точки. A * врівноважує їх у міру просування від початкової точки до мети. Кожен раз, проходячи через основний цикл, він перевіряє вершину n з найменшим f (n) = g (n) + h (n).

Обґрунтування методу розв'язання

Базуючись на інформації представленій в розділі 3.3 алгоритм пошуку шляху A* є най оптимальнішим з відомих варіантів. В додаток до вже існуючих переваг випраного алгоритму його результативність можливо підвищити за рахунок оптимізацій наведених далі.

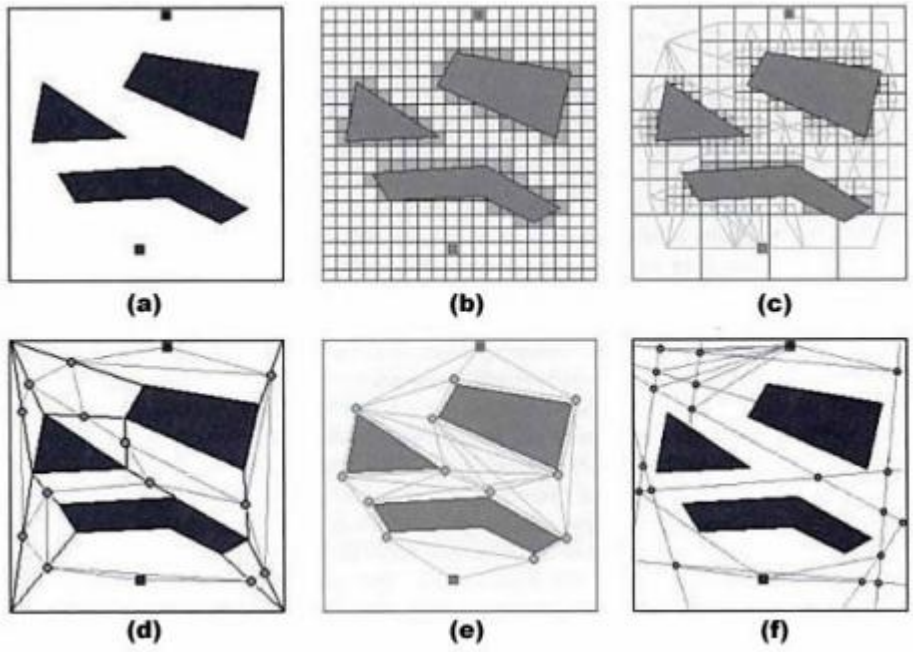
Простір пошуку

У будь-ігровому середовищі AI-персонажі повинні використовувати базова структура даних - представлення простору пошуку - планувати шлях до будь-якого заданого місця призначення. знаходження найбільш підходяща структура даних для представлення пошуку простір для ігрового світу абсолютно необхідно для досягнення реалістично виглядає руху і прийнятного пошук шляху продуктивності. Як ви можете бачити в наведеному вище Наприклад, більш просте простір пошуку означатиме, що A * має менше роботи і менше роботи дозволять алгоритму біжи швидше.

У наступних підрозділах розглядаються два популярних A*-based алгоритми, які оптимізують алгоритм A* шляхом зменшення простору пошуку.

Ієрархічний пошук шляху A* (HPA*)

Ієрархічний пошук шляхів - це надзвичайно потужний метод, який прискорює процес пошуку шляхів. Складність проблеми може бути зменшена шляхом ієрархічного поділу світу.



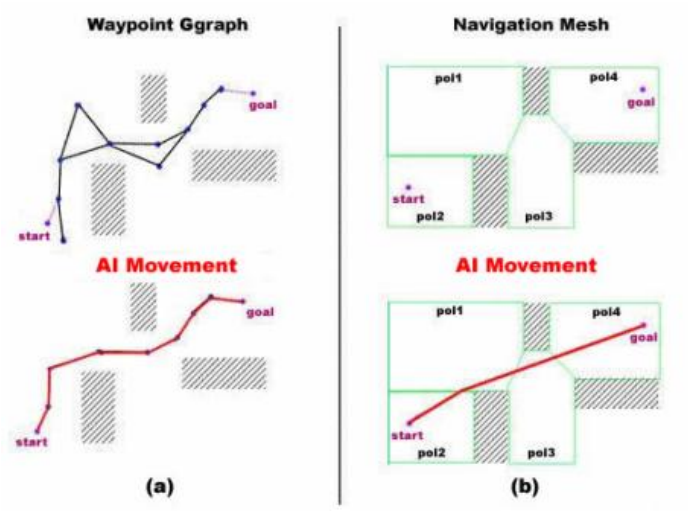
(а) – сітка, (в) – вадро, (г) – опуклі полігони, (д) – точки видимості і (f) – узагальнені циліндри.

Рисунок 3.8 – П'ять способів представити простір пошуку.

Набагато більш швидкий алгоритм пошуку на основі A *, що дає майже рішення під назвою HPA*. Це незалежний від домену підхід. Ієрархія може бути розширена до більш ніж двох рівнів, що робить її більш масштабованої для великих проблемних просторів. Триступеневий процес застосовується. Перший крок - перейти до кордону району, в якому знаходиться початкова розташування. Потім другим кроком є пошук шляху від кордону стартовою околиць аж до кордону цільової околиці. Цей крок виконується на абстрактному рівні, де пошук простіше і швидше. Останній крок - завершити шлях, пройшовши від кордону району мети до позиції мети. Потенційна проблема цього методу полягає в тому, що вартість значно збільшується при додаванні нового рівня абстракції.

Сітка навігації (NavMesh)

NavMesh - ще одна популярна техніка для пошуку AI в 3D-світи. NavMesh - це набір опуклих багатокутників, які описують «прохідну» поверхню тривимірної середовища. Це простий, інтуїтивно зрозумілий план поверху, який AI-персонажі можуть використовувати для навігації і пошуку шляхів в ігровому світі. На нижче наведеному показаний приклад NavMesh. Символ переміщається з початкової точки в pol2 до бажаного місця призначення в pol4. У цьому випадку початкова точка знаходиться не в тому ж багатокутнику, що і бажана точка. Таким чином, персонаж повинен визначити наступний полігон, на який він піде. Повторюйте цей крок, поки персонаж і мета не опиняться в одному і тому ж багатокутнику. Потім персонаж може рухатися до місця призначення по прямій лінії.



(а) – графу точок та (б) – NavMesh.

Рисунок 3.9 – Графічне представлення.

У порівнянні з графіком шляхових точок, показаним на рис. 10, підхід NavMesh гарантує пошук майже оптимального шляху шляхом пошуку значно меншої кількості даних. І поведінка пошуку шляху в NavMesh перевершує поведінку в графі шляхових точок .

Висновок до розділу

У цій статті систематично розглядаються кілька популярних A* Алгоритми і методи на основі оптимізація A*. Це показує чітко реляційну карту між алгоритмом A* і його варіантами. Ядро Алгоритм пошуку шляху - тільки маленький шматочок головоломки в грі AI. Найскладніше завдання полягає в тому, як використовувати алгоритм вирішення складних завдань. A* алгоритм є найпопулярніший алгоритм в пошуку шляху. Це важко знайти кращий алгоритм, оскільки A* доказово оптимально. багато зусиль було вкладено в його прискорення оптимізувати його з різних точок зору. Способи поліпшити продуктивність пошуку A* оптимізувати базове простір пошуку, зменшивши використання пам'яті, поліпшення евристичних функцій і впровадження нових структур даних.

ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

Засоби розробки

Для реалізації програмного продукту було проаналізовано різні системи для створення ігор (Game engine) , щоб визначити який з них найкраще підійде для вирішення поставлених задач та досягнення мети.

Після аналізу наявних систем було вирішено розробляти гру за допомогою програми Unreal Engine 4.

Unreal Engine - це вичерпний набір інструментів розробки для тих, хто працює з технологіями комп'ютерних симуляцій в реального часу. Від візуалізації дизайну та кінематографічного досвіду до високоякісних ігор на ПК, консолі, мобільних пристроях, VR та AR, Unreal Engine надає вам все необхідне для запуску, доставки, росту та вирішення проблем серед натовпу[15].

UE підтримує розробку під різні платформи від ПК до консолей, таких як Xbox One, PS4 та Switch. Це є причиною того, що він настільки широко використовується, завдяки своїй гнучкості роботи між цими різними платформами.

Розробники можуть використовувати мову програмування C++ для створення власних сценаріїв, що працюють на базі системи UE. Розробники, що не знайомі з C++, можуть використовувати візуальну мову програмування blueprint, яку ви можете додати до своїх об'єктів для взаємодії.

Великою перевагою Unreal Engine є те, що він абсолютно безкоштовний у використанні. Незалежно від того, що ви хобі-розробник чи студія AAA, за використання Unreal не існує додаткової плати. Вони замість цього використовують угоду про роялті в розмірі 5% від усіх доходів від гри, що перевищують 3000 доларів на квартал.

Вимоги до технічного забезпечення

Системні вимоги для середовища розробки UE4

- Операційна система: Windows 7/8 64-біт;
- Процесор: Quad-core Intel або AMD, 2.5 GHz або вище;
- Пам'ять (ОЗП): 8 Гб;
- Відеокарта / версія DirectX: відеокарта сумісна з DirectX 11;

Системні вимоги для гри на різних платформах

PC:

- ОС Windows 7/8 / 8.1 / 10 (64-розрядна);
- процесор Intel Celeron G1820 | AMD A4-7300;
- пам'ять4- ГБ оперативної пам'яті;
- відеокарта NVIDIA GeForce GT 730 | Radeon R7 240;
- Дозвіл екрану: 1280 x 720 або вище;
- Мережа: швидкісний доступ в інтернет;
- Вільне місце на диску: 6 ГБ;
- Звукова карта: сумісна з DirectX.

Android:

- OS: Версія ОС: Android 4.2 і вище;
- Відеочіпи не слабкіше: Mali-400MP, PowerVR SGX544, Adreno 320;
- Процесор: від 1200 МГц (рекомендується 1500 МГц), 2 ядра;
- Оперативна пам'ять: 1 ГБ і вище.

iOS

- Версія ОС: iOS 9 і вище;
- Пристрої: iPhone 5 / iPad 3 / iPad mini 2 / iPod touch 6G і новіше.

Архітектура програмного забезпечення

Так як система розробляється за допомогою Unreal Engine 4 вона базується на його архітектурі і розширює її.

Таким же чином, як сам UE4 складається з набору модулів, кожна гра складається з одного або декількох ігрових модулів. Вони схожі на пакети в попередніх версіях UE4 тим, що вони є контейнерами для колекції пов'язаних класів. В Unreal Engine 4, оскільки весь ігровий процес виконується на C++, модулі насправді є DLL-файлами, а не приватними пакетними файлами.

При програмуванні елементів ігрового процесу з використанням коду C++ кожен модуль може містити безліч класів C++ (рис. 4.1).

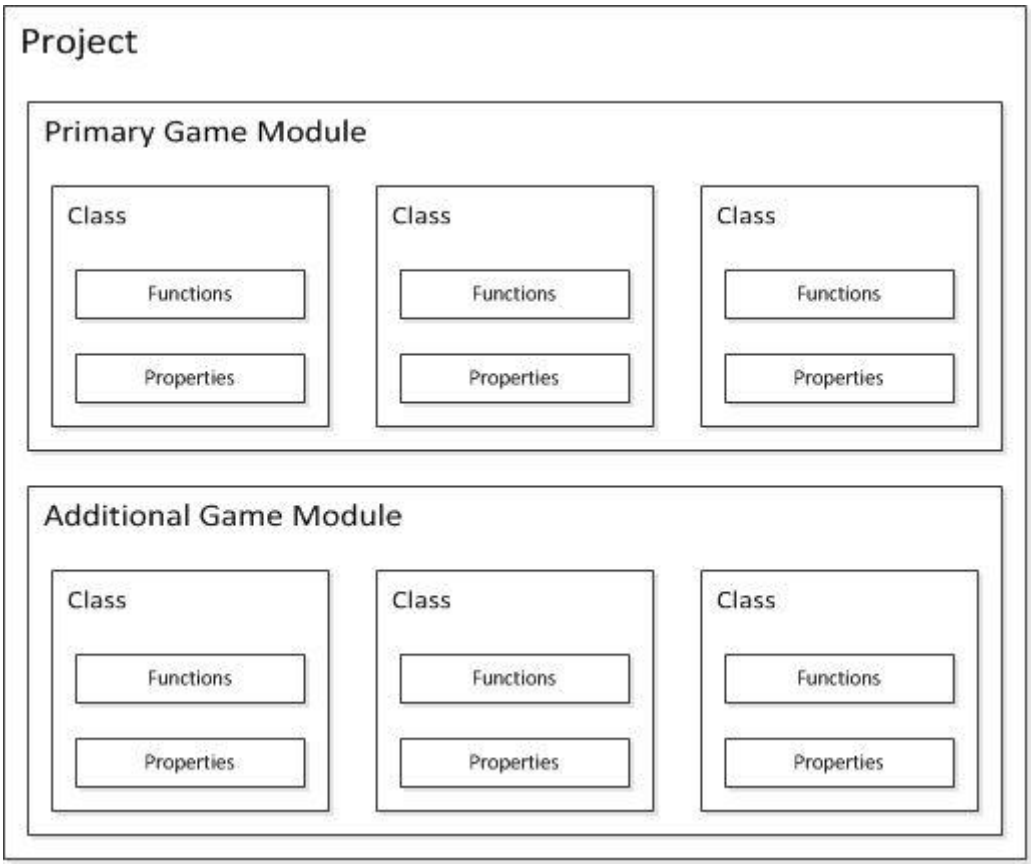


Рисунок 3 – Схема архітектури програмного забезпечення.

Кожен клас визначає шаблон для нового Actor або Object. В заголовки класу оголошуються клас і всі функції і властивості класу. Класи також можуть містити структури, структури даних, які допомагають в організації та

управлінні пов'язаними властивостями. Структури також можуть бути визначені самостійно. Інтерфейс дозволяє реалізовувати додаткове ігрове поведінку різними класами.

При програмуванні з Unreal Engine можливо мати стандартні класи, функції і змінні C ++. Вони можуть бути визначені з використанням стандартного синтаксису C ++. Однак макроси UCLASS (), UFUNCTION () і UPROPERTY () можна використовувати для інформування Unreal Engine про нові класах, функціях і змінних. Наприклад, змінна з оголошенням, яка випереджає макросом UPROPERTY (), може бути оброблена складальником сміття і може відображатися і редагуватися в Unreal Editor. Існують також макроси UINTERFACE () і USTRUCT (), а також ключові слова для кожного макросу, які можна використовувати для вказівки поведінки класу, функцій, властивостей, інтерфейсу або структури в Unreal Engine і Unreal Editor.

На додаток до вищезазначених макросам є макрос UPARAM (), який в основному використовується при поданні коду C ++ для Blueprints.

Діаграма класів

Основні класи ігрового процесу включають в себе функціональність для подання гравців, союзників і ворогів, а також для управління цими аватарами за допомогою входів гравця або логіки AI. Є також класи для створення хедз-ап дисплеїв і камер для гравців. Нарешті, ігрові класи, такі як GameMode, GameState і PlayerState, встановлюють правила гри і відстежують хід гри і гравців. Гравці, що приєднуються до гри, взаємодіють з грою через PlayerControllers (рис. 4.2).

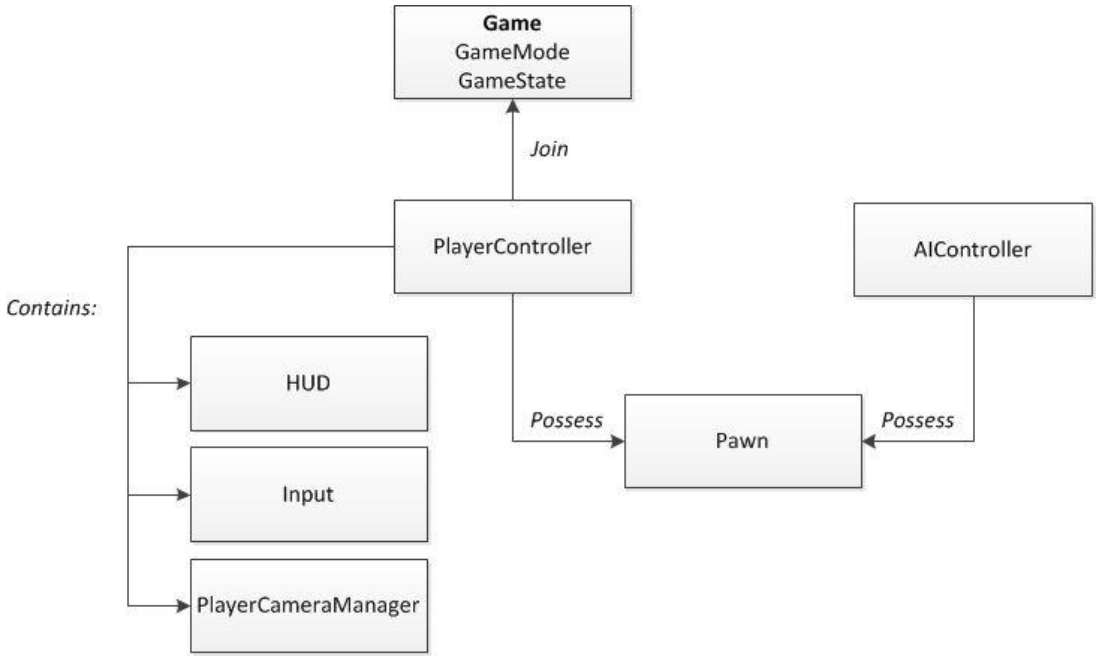


Рисунок 4 – Діаграма класів.

PlayerControllers дозволяють гравцям мати підконтрольних їм агентів у грі, щоб вони могли мати фізичне представлення на рівні. PlayerControllers також надають гравцям елементи управління введенням, екран з написом або HUD і PlayerCameraManager для обробки зображень з камер.

Опис задіяних класів

Actor (Актори) - є екземплярами класів, які є похідними від класу AActor; базовий клас всіх ігрових об'єктів, які можуть бути розміщені в світі. Об'єкти є екземплярами класів, які успадковуються від класу UObject; базовий клас всіх об'єктів в Unreal Engine, включаючи акторів. Таким чином, в дійсності все об’єкти у Unreal Engine є об'єктами; проте термін Actors зазвичай використовується для позначення примірників класів, які є похідними від AActor в їх ієрархії, в той час як термін Objects використовується для посилання на екземпляри класів, які не успадковуються від класу AActor. Більшість створюваних вами класів будуть успадковуватися від AActor в якийсь момент їх ієрархії.

Pawn (пішак) - це Актор, який може бути «агентом» в світі. Контролером можуть володіти пішаки, вони налаштовані так, щоб легко приймати вхідні дані, і вони можуть робити різні і інші речі, схожі на гравців. Зверніть увагу, що пішак не рахується гуманоїдом.

Character (персонаж) - Пішак в гуманоїдні стилі. Він поставляється з CapsuleComponent для зіткнення і CharacterMovementComponent за замовчуванням. Він може виконувати базові рухи, подібні до людських, він може плавно копіювати рух по мережі і має деякі функції, пов'язані з анімацією.

Controller (контролер) - це Актор, який відповідає за управління Пішаком. Вони зазвичай бувають двох видів: AIController і PlayerController.

PlayerController - це інтерфейс між Пішаком і гравцем, який контролює її. PlayerController по суті являє волю людини-гравця.

AIController – як слідує з назви це симулювати воля, яка може контролювати пішака.

HUD - це «екранний дисплей», або 2D-екран, широко поширений в багатьох іграх. Той елемент де відображається здоров'я, боєприпасах, приціл зброї і тому подібне.

PlayerCameraManager є «очима» гравця і керує тим, що бачить гравець. Кожен PlayerController зазвичай контролює один з відповідних екземплярів цього класу.

GameState містить стан гри, який може включати в себе такі речі, як список підключених гравців, рахунок, кількість фігур в шаховій грі або список місій, які ви виконали в грі з відкритим світом. GameState існує на сервері і на всіх клієнтах і може вільно копіювати, щоб підтримувати всі машини в актуальному стані.

PlayerState - це стан учасника гри, такого як гравець або бот, що імітує гравця. Приблизні дані, які були б доречні в PlayerState, включають ім'я гравця, рахунок, рівень в матчі. PlayerStates для всіх гравців існують на всіх

машинах (на відміну від PlayerControllers) і можуть вільно реплікуватися для забезпечення синхронізації.

Діаграма послідовності

На рисунку 4.3 зображено діаграму послідовності для розробленого застосунку.

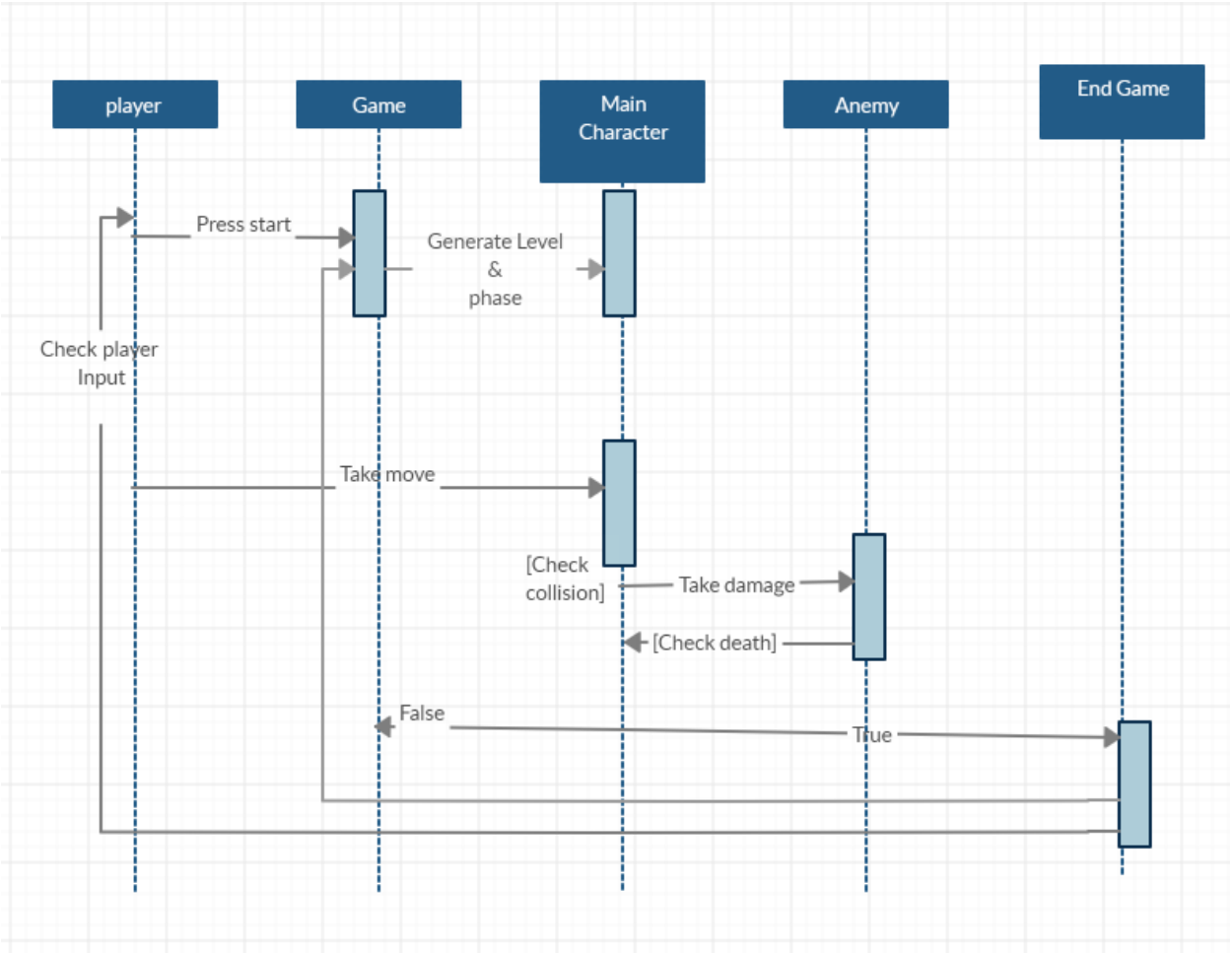


Рисунок 4.3 – Діаграма послідовності.

Діаграма компонентів

На рисунку 4.4 зображено діаграму компонентів для розробленого застосунку.

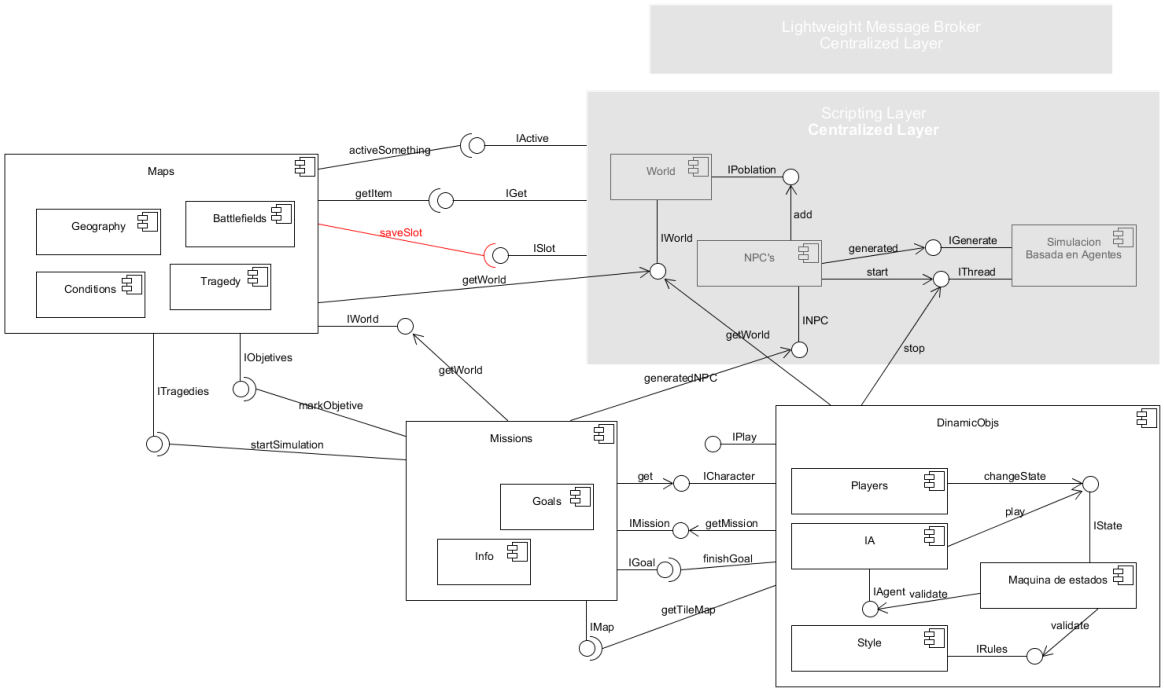


Рисунок 4.4 – Діаграма компонентів.

Специфікація функцій

Таблиця 4.1 – Таблиця специфікації функцій.

Назва функції	**Опис функції**
AIMoveTo	Простий наказ для пішака з AIController , про рух до конкретної точки
ClearFocus	Очищає фокус, також очищає FocalPoint в результаті. Метою є AIController

GetAvoidanceVelocity	Розрахувати швидкість ухилення для компонента (уникає зіткнень з поставляється компонентом)
GetCurrentPath	Повертає копію навігаційного шляху, який даний контролер використовує. У результаті копія означає, що ви не зможете впливати на переміщення агента через маніпулювання отриманим шляхом
GetFocalPoint	Повертає остаточне положення, на яке повинен дивитися контролер.
GetFocalPoint	Отримайте фокус, на якому цей контролер повинен зосередити увагу
MakeNoise	Запустіть шум, викликаний певною пішаком, у заданому місці. Відправники MakeNoise повинні мати Instigator, якщо вони не пішаки, або передають NoiseInstigator.
SpawnAIFromClass	Породжує AI агента даного класу. PawnClass повинен мати AIController для функції створення контролера.

UseBlackboard	Змушує AI використовувати вказаний актив Blackboard і створює компонент Blackboard, якщо такий ще не існує.

Опис звітів

До створеною системи звітом є Log-файл в якому збережено данні про роботу системи. За допомогою його файлу можна отримати повну картину того що відбувалось під час роботи системи.

Також система логування в UE4 підтримую гнучке налаштування вихідного файлу, що дозволяє полегшити і підвищити продуктивність роботи з ним.

Одним з таких інструментів є LogVerbosityLevels. LogVerbosityLevels використовується для більш легкого контролю надрукованих даних, що дозволяє зберігати навіть найдокладніші log-виписки у своєму коді, не перетворюючи їх на спам, коли ви цього їх не використовуєте. Кожен оператор логування оголошує, якому журналу він належить, і це є рівнем логування. Багаторівневість кожного журналу контролюється чотирма речами: рівнем заданим на етапі компіляції, рівнем за замовчуванням, налаштуваннями з ini-файлу та рівнем заданим під час роботи програми.

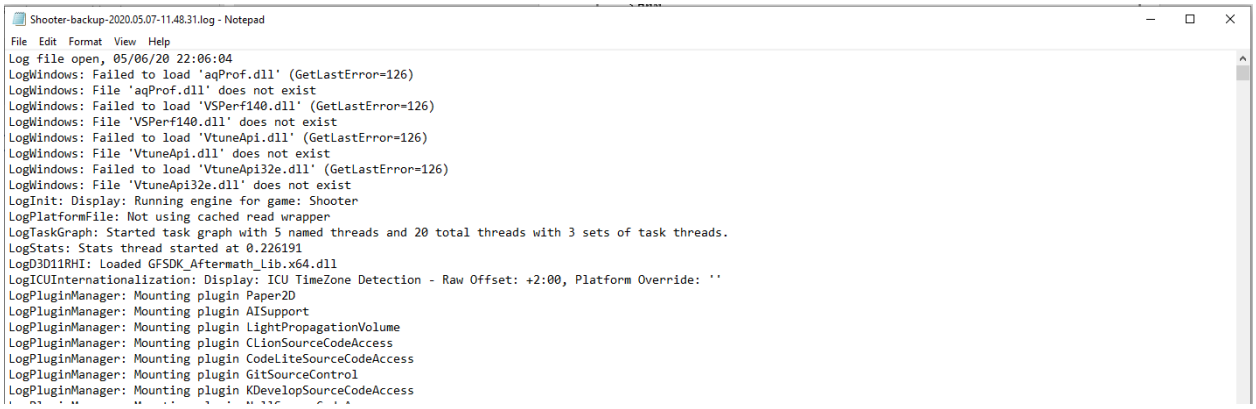


Рисунок 4.5 – Приклад Log файлу.

Висновок до розділу

У процесі написання даного розділу було розглянуто засоби розробки програмного забезпечення, наведено характеристику системи на базі якої був розроблений програмний продукт.

Були виявлені загальні вимоги до програмного та технічного забезпечення, необхідних для роботи програмного продукту.

Також було детально описано архітектуру архітектурну частину програмного забезпечення. Створено діаграми класів, послідовності та компонентів. Надано специфікацію функцій, які використовувалися підчас створення програмного продукту і описано звіти, які генеруються в результаті роботи програми.

ТЕХНОЛОГІЧНИЙ РОЗДІЛ

Керівництво користувача

Оскільки кінцевим продуктом є відео гра в цьому розділі буде наведено керівництво користувача який виступає в ролі гравця.

Гра представляє варіацію доволі відомого типу twin stick shooter. Основними рисами цього жанру є те що гравець управляє своїм персонажем за допомогою двох стіків. Один стік відповідає за переміщення персонажа, а інший за стрільбу. У випадку розробленого продукту лівий стік відповідає за переміщення, а правий за постріли. В залежності від платформи це можуть бути:

- фізичні стіки геймпаду у випадку персонального комп’ютера або консолі (рис 5.1);

Рисунок 5.1 – Зображення геймпаду.

- віртуальні стіки на екрані пристрою у випадку смартфону або планшету (рис 5.2).

Рисунок 5.2 – Зображення смартфону із віртуальними стіками.

Головною задачею гри є знешкодження ворожих агентів якими керує розроблена система. Гра закінчується коли ворожим агентам вдається знешкодити персонаж гравця. Персонаж гравця пофарбований у білий колір а ворожі агенти у червоний. Також для зручності гри на ігровий екран винесено інформацію про запас міцності персонажа гравця (HP) і його ігровий рахунок (Score).

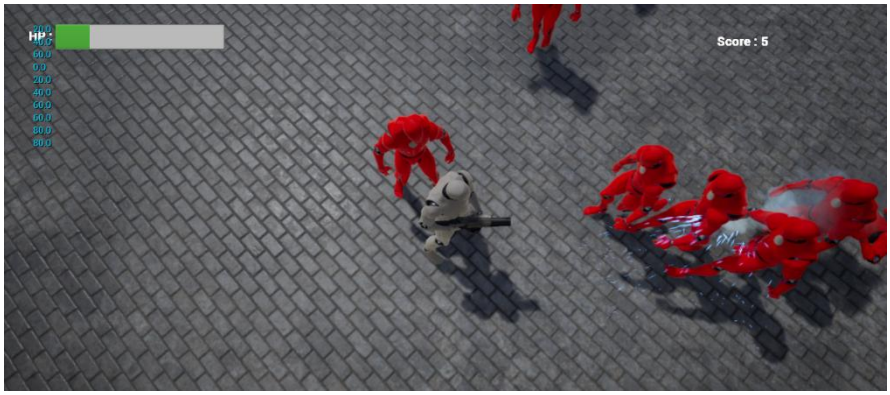


Рисунок 5.3 – Скріншот програмного продукту.

Випробування програмного продукту

Мета випробувань

Метою випробувань являється перевірка відповідності функцій комплексу задач управління агентами у відео грі вимогам технічного

завдання.

Для досягнення мети потрібно провести функціональне тестування готової збірки гри і переконатись в тому, що система видає очікуваний від неї результат в кожному тестів.

Загальні положення

Випробування проводяться на основі наступних документів:

- ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
- ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

Результати випробувань

В процесі тестування була перевірена уся функціональність системи. Далі наведено перелік функціональних тестів через які пройшла система.

Створення персонажу гравця. Персонаж повинен конструюватись та розміщуватись в ігровому світі у двох сценаріях.

- при першому запуску гри;
- після знешкодження персонажу гравця ворожими агентами.

Було проведено декілька запусків програми та змодельовано декілька ситуацій знешкодження персонажу гравця. У всіх ітераціях тесту програма успішно створювала ігрового персонажа.

Переміщення. Ігровий персонаж повинен переміщуватись у напрямку заданим лівим стіком. Ситуації які моделювались при проходженні ітерацій тесту:

- переміщення одразу після створення гравця;
- переміщення після зупинки ігрового персонажу;
- переміщення у різних точках ігрової локації.

Постріл. При зміщенні правого стіку від центру персонаж повинен повертатись у відповідному напрямку та здійснювати постріл. Було по ведено тести зміщення стіка відносно вихідного положення спокою, а також зміщення стіку відносно попередньої позиції. У всіх випадках персонаж повертався у напрямі заданому стікою та відбувався постріл.

Ураження ворожого агента кульою. Для проведення цього тесту було додано додаткову дебажну інформацію, яка виводилася на ігровий екран і відображала значення мігності ворожого агента після колізії з кульою.

Висновок до розділу

У процесі написання цього розділу, було сформовано керівництво користувача та наведено екранні форми розробленої застосунку.

Також встановлено мету проведення випробувань, наведено загальні положення. Визначено тести які необхідно провести для валідації різних складових застосунку.

У процесі тестування було перевірено усю функціональність системи, наведений перелік випробувань основних функціональних можливостей.

У процесі тестування програмного продукту встановлено, що функціонал розробленого застосунок відповідає встановленим вимогам.

ЗАГАЛЬНІ ВИСНОВКИ

Основна функція Штучного інтелекту (AI) у виробництві ігор має виграшне значення і часто впливає на успіх чи невдачу гри. Але AI, використовуваний у комп'ютерних іграх, не повинен бути штучним інтелектом в повному розумінні цього терміну. Цілі ігрового AI спрямовані на прогрес та покращення досвіду гри.

Під час першого етапу роботи над дипломним проектом було визначено актуальність обраної теми та досліджено засоби, що розроблялись раніше для вирішення схожих задач.

Було детально описано процес діяльності, що автоматизується, визначено, які саме етапи необхідно пройти, для розв'язання задачі, представлено характеристику методів, за допомогою яких пропонується вирішувати поставлену задачу. Було розроблено діаграму діяльності та наведено її опис.

Також, було описано функціональну модель системи та визначено функції користувача. Сформульовано призначення системи що розробляється, визначено мету і задачі, що потрібно вирішити для досягнення мети.

Під час наступного етапу було визначено вхідні та вихідні дані системи, яка розробляється.

Представлено опис вхідних даних задачі, було формалізовано саму задачу та наведено основні атрибути задачі. Встановлено математичне описання задачі. Наведено опис способів розв'язання задачі, наведено детальну характеристику кожного алгоритму. Встановлено недоліки та переваги алгоритмів.

Також були виявлені загальні вимоги до програмного продукту, описано рішення, які використовуватимуться під час розробки програмного забезпечення.

Було описано архітектуру модель програмного продукту. Створено діаграми класів, послідовності та компонентів і наведено їх опис.

Також представлено специфікацію функцій, які використовувалися при створенні програмного продукту та описано звіти, що генеруються в результаті роботи програми.

У процесі тестування була перевірена вся функціональність системи, наведений перелік випробувань основних функціональних можливостей.

Використовуючи розроблену систему управління агентами кожен може створити відео гру на основі обраних алгоритмів для вирішення поставленої задачі.

Отже, цілі, що були поставлені на початковому етапі проектування системи управління агентами у відео грі були досягненими в ході виконання дипломного проекту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bourg; Seemann (2004). AI for Game Developers. O'Reilly & Associates. ISBN 0-596-00555-5.
2. Buckland (2002). AI Techniques for Game Programming. Muska & Lipman. ISBN 1-931841-08-X.
3. Buckland (2004). Programming Game AI By Example. Wordware Publishing. ISBN 1-55622-078-2.
4. Champandard (2003). AI Game Development. New Riders. ISBN 1-59273-004-3.
5. Funge (1999). AI for Animation and Games: A Cognitive Modeling Approach. A K Peters. ISBN 1-56881-103-9.
6. Funge (2004). Artificial Intelligence for Computer Games: An Introduction. A K Peters. ISBN 1-56881-208-6.
7. Millington (2005). Artificial Intelligence for Games. Morgan Kaufman. ISBN 0-12-497782-0.* Schwab (2004). AI Game Engine Programming. Charles River Media. ISBN 1-58450-344-0.
8. Smed and Hakonen (2006). Algorithms and Networking for Computer Games. John Wiley & Sons. ISBN 0-470-01812-7.
9. Lara-Cabrera, R. Game artificial intelligence: Challenges for the scientific community / R. Lara-Cabrera, M. Nogueira-Collazo. // CEUR Workshop Proceedings. – 2015. – №1394. – С. 1-12.
10. Yannakakis, Geogios N. Game AI revisited / N. Yannakakis, Geogios. // Proceedings of the 9th conference on Computing Frontiers. – 2012. – С. 285–292
11. Fullerton T. Game Design Workshop / T. Fullerton – Morgan Kaufmann, 2008. – С.
12. Data Driven Gameplay Elements. https://docs.unrealengine.com/en-US/Gameplay/DataDriven/index.html.

13.A* Search. https://brilliant.org/wiki/a-star-search/

14.Unreal Engine 4 AI Programming Essentials. By Peter L. Newton, Jie Feng. March 2016

15.Unreal Engine Features https://www.unrealengine.com/en-US/features

Додаток А

Тексти програмного коду

Система управління агентами у відео грі

(Найменування програми (документа))

DVD-R

(Вид носія даних)

10 арк, 244 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2020 року

```
#include "BattleTank.h"
#include "Projectile.h"


// Sets default values
AProjectile::AProjectile()
{
	// Set this actor to call Tick() every frame.  You can turn this off
to improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = false;

	CollisionMesh =
CreateDefaultSubobject<UStaticMeshComponent>(FName("Collision Mesh"));
	SetRootComponent(CollisionMesh);
	CollisionMesh->SetNotifyRigidBodyCollision(true);
	CollisionMesh->SetVisibility(false);

	LaunchBlast =
CreateDefaultSubobject<UParticleSystemComponent>(FName("Launch Blast"));
	LaunchBlast->AttachToComponent(RootComponent,
FAttachmentTransformRules::KeepRelativeTransform);

	ProjectileMovement =
CreateDefaultSubobject<UProjectileMovementComponent>(FName("Projectile
Movement"));
	ProjectileMovement->bAutoActivate = false;

	ImpactBlast =
CreateDefaultSubobject<UParticleSystemComponent>(FName("Impact Blast"));
	ImpactBlast->AttachToComponent(RootComponent,
FAttachmentTransformRules::KeepRelativeTransform);
	ImpactBlast->bAutoActivate = false;

	ExplosionForce =
CreateDefaultSubobject<URadialForceComponent>(FName("Explosion Force"));
	ExplosionForce->AttachToComponent(RootComponent,
FAttachmentTransformRules::KeepRelativeTransform);
}

// Called when the game starts or when spawned
void AProjectile::BeginPlay()
{
	Super::BeginPlay();
	CollisionMesh->OnComponentHit.AddDynamic(this, &AProjectile::OnHit);
}

void AProjectile::LaunchProjectile(float Speed)
{
	ProjectileMovement->SetVelocityInLocalSpace(FVector::ForwardVector *
Speed);
	ProjectileMovement->Activate();
}

void AProjectile::OnHit(UPrimitiveComponent* HitComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComponent, FVector NormalImpulse, const
FHitResult& Hit)
{
	LaunchBlast->Deactivate();
	ImpactBlast->Activate();
```

```
	ExplosionForce->FireImpulse();

	SetRootComponent(ImpactBlast);
	CollisionMesh->DestroyComponent();

	UGameplayStatics::ApplyRadialDamage(
		this,
		ProjectileDamage,
		GetActorLocation(),
		ExplosionForce->Radius, // for consistancy
		UDamageType::StaticClass(),
		TArray<AActor*>() // damage all actors
	);

	FTimerHandle Timer;
	GetWorld()->GetTimerManager().SetTimer(Timer, this,
&AProjectile::OnTimerExpire, DestroyDelay, false);
}

void AProjectile::OnTimerExpire()
{
	Destroy();
}




#include "BattleTank.h"
#include "SpawnPoint.h"


// Sets default values for this component's properties
USpawnPoint::USpawnPoint()
{
	// Set this component to be initialized when the game starts, and to
be ticked every frame.  You can turn these features
	// off to improve performance if you don't need them.
	PrimaryComponentTick.bCanEverTick = true;

	// ...
}


// Called when the game starts
void USpawnPoint::BeginPlay()
{
	Super::BeginPlay();

	SpawnedActor = GetWorld()->SpawnActorDeferred<AActor>(SpawnClass,
GetComponentTransform());
	if (!SpawnedActor) return;
	SpawnedActor->AttachToComponent(this,
FAttachmentTransformRules::KeepWorldTransform);
	UGameplayStatics::FinishSpawningActor(SpawnedActor,
GetComponentTransform());
}


// Called every frame
void USpawnPoint::TickComponent(float DeltaTime, ELevelTick TickType,
FActorComponentTickFunction* ThisTickFunction)
```

```
{
	Super::TickComponent(DeltaTime, TickType, ThisTickFunction);

	// ...
}



#include "BattleTank.h"
#include "SprungWheel.h"

#include "PhysicsEngine/PhysicsConstraintComponent.h"

// Sets default values
ASprungWheel::ASprungWheel()
{
	// Set this actor to call Tick() every frame.  You can turn this off
to improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;
	PrimaryActorTick.TickGroup = TG_PostPhysics;

	MassWheelConstraint =
CreateDefaultSubobject<UPhysicsConstraintComponent>(FName("MassWheelConstrain
t"));
	SetRootComponent(MassWheelConstraint);

	Axle = CreateDefaultSubobject<USphereComponent>(FName("Axle"));
	Axle->SetupAttachment(MassWheelConstraint);

	Wheel = CreateDefaultSubobject<USphereComponent>(FName("Wheel"));
	Wheel->SetupAttachment(Axle);

	AxleWheelConstraint =
CreateDefaultSubobject<UPhysicsConstraintComponent>(FName("AxleWheelConstrain
t"));
	AxleWheelConstraint->SetupAttachment(Axle);

}

// Called when the game starts or when spawned
void ASprungWheel::BeginPlay()
{
	Super::BeginPlay();

	Wheel->SetNotifyRigidBodyCollision(true);
	Wheel->OnComponentHit.AddDynamic(this, &ASprungWheel::OnHit);

	SetupConstraint();
}

void ASprungWheel::SetupConstraint()
{
	if (!GetAttachParentActor()) return;
	UPrimitiveComponent* BodyRoot =
Cast<UPrimitiveComponent>(GetAttachParentActor()->GetRootComponent());
	if (!BodyRoot) return;
	MassWheelConstraint->SetConstrainedComponents(BodyRoot, NAME_None,
Axle, NAME_None);
	AxleWheelConstraint->SetConstrainedComponents(Axle, NAME_None, Wheel,
NAME_None);
```

```
}

// Called every frame
void ASprungWheel::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	if (GetWorld()->TickGroup == TG_PostPhysics)
	{
		TotalForceMagnitudeThisFrame = 0;
	}
}

void ASprungWheel::AddDrivingForce(float ForceMagnitude)
{
	TotalForceMagnitudeThisFrame += ForceMagnitude;
}

void ASprungWheel::OnHit(UPrimitiveComponent* HitComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp, FVector NormalImpulse, const
FHitResult& Hit)
{
	ApplyForce();
}

void ASprungWheel::ApplyForce()
{
	Wheel->AddForce(Axle->GetForwardVector() *
TotalForceMagnitudeThisFrame);
}


#include "BattleTank.h"
#include "Tank.h"


float ATank::GetHealthPercent() const
{
	return (float)CurrentHealth / (float)StartingHealth;
}

// Sets default values
ATank::ATank()
{
	// Set this pawn to call Tick() every frame.  You can turn this off to
improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = false;
}

void ATank::BeginPlay()
{
	Super::BeginPlay();
	CurrentHealth = StartingHealth;
}

float ATank::TakeDamage(float DamageAmount, struct FDamageEvent const &
DamageEvent, class AController * EventInstigator, AActor * DamageCauser)
{
	int32 DamagePoints = FPlatformMath::RoundToInt(DamageAmount);
	int32 DamageToApply = FMath::Clamp(DamagePoints, 0, CurrentHealth);
```

```
	CurrentHealth -= DamageToApply;
	if (CurrentHealth <= 0)
	{
		OnDeath.Broadcast();
	}
	return DamageToApply;
}


#include "BattleTank.h"
#include "TankAimingComponent.h"
#include "TankAIController.h"
#include "Tank.h" // So we can impliment OnDeath

// Depends on movement component via pathfinding system

void ATankAIController::BeginPlay()
{
	Super::BeginPlay();
}

void ATankAIController::SetPawn(APawn* InPawn)
{
	Super::SetPawn(InPawn);
	if (InPawn)
	{
		auto PossessedTank = Cast<ATank>(InPawn);
		if (!PossessedTank) { return; }

		// Subscribe our local method to the tank's death event
		PossessedTank->OnDeath.AddUniqueDynamic(this,
&ATankAIController::OnPossedTankDeath);
	}
}

void ATankAIController::OnPossedTankDeath()
{
	if (!ensure(GetPawn())) { return; } // TODO remove if ok
	GetPawn()->DetachFromControllerPendingDestroy();
}

// Called every frame
void ATankAIController::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	auto PlayerTank = GetWorld()->GetFirstPlayerController()->GetPawn();
	auto ControlledTank = GetPawn();

	if (!(PlayerTank && ControlledTank)) { return; }

	// Move towards the player
	MoveToActor(PlayerTank, AcceptanceRadius); // TODO check radius is in
cm

	// Aim towards the player
	auto AimingComponent = ControlledTank-
>FindComponentByClass<UTankAimingComponent>();
	AimingComponent->AimAt(PlayerTank->GetActorLocation());

	if (AimingComponent->GetFiringState() == EFiringState::Locked)
```

```
	{
		AimingComponent->Fire(); // TODO limit firing rate
	}
}



#include "BattleTank.h"
#include "TankBarrel.h"
#include "TankTurret.h"
#include "Projectile.h"
#include "TankAimingComponent.h"


// Sets default values for this component's properties
UTankAimingComponent::UTankAimingComponent()
{
	// Set this component to be initialized when the game starts, and to
be ticked every frame.  You can turn these features
	// off to improve performance if you don't need them.
	PrimaryComponentTick.bCanEverTick = true;
}

void UTankAimingComponent::BeginPlay()
{
	Super::BeginPlay();
	// So that first first is after initial reload
	LastFireTime = FPlatformTime::Seconds();
}

void UTankAimingComponent::Initialise(UTankBarrel* BarrelToSet, UTankTurret*
TurretToSet)
{
	Barrel = BarrelToSet;
	Turret = TurretToSet;
}

void UTankAimingComponent::TickComponent(float DeltaTime, enum ELevelTick
TickType, FActorComponentTickFunction *ThisTickFunction)
{
	if (RoundsLeft <= 0)
	{
		FiringState = EFiringState::OutOfAmmo;
	}
	else if ((FPlatformTime::Seconds() - LastFireTime) <
ReloadTimeInSeconds)
	{
		FiringState = EFiringState::Reloading;
	}
	else if (IsBarrelMoving())
	{
		FiringState = EFiringState::Aiming;
	}
	else
	{
		FiringState = EFiringState::Locked;
	}
}

int32 UTankAimingComponent::GetRoundsLeft() const
```

```
{
	return RoundsLeft;
}

EFiringState UTankAimingComponent::GetFiringState() const
{
	return FiringState;
}


bool UTankAimingComponent::IsBarrelMoving()
{
	if (!ensure(Barrel)) { return false; }
	auto BarrelForward = Barrel->GetForwardVector();
	return !BarrelForward.Equals(AimDirection, 0.01); // vectors are equal
}

void UTankAimingComponent::AimAt(FVector HitLocation)
{
	if (!ensure(Barrel)) { return; }

	FVector OutLaunchVelocity;
	FVector StartLocation = Barrel-
>GetSocketLocation(FName("Projectile"));
	bool bHaveAimSolution = UGameplayStatics::SuggestProjectileVelocity
	(
		this,
		OutLaunchVelocity,
		StartLocation,
		HitLocation,
		LaunchSpeed,
		false,
		0,
		0,
		ESuggestProjVelocityTraceOption::DoNotTrace // Paramater must
be present to prevent bug
	);

	if (bHaveAimSolution)
	{
		AimDirection = OutLaunchVelocity.GetSafeNormal();
		MoveBarrelTowards(AimDirection);
	}
	// If no solution found do nothing
}

void UTankAimingComponent::MoveBarrelTowards(FVector TargetAimDirection)
{
	if (!ensure(Barrel) || !ensure(Turret)) { return; }

	// Work-out difference between current barrel roation, and
AimDirection
	auto BarrelRotator = Barrel->GetForwardVector().Rotation();
	auto AimAsRotator = TargetAimDirection.Rotation();
	auto DeltaRotator = AimAsRotator - BarrelRotator;

	// Always yaw the shortest way
	Barrel->Elevate(DeltaRotator.Pitch);
	if (FMath::Abs(DeltaRotator.Yaw) < 180)
	{
		Turret->Rotate(DeltaRotator.Yaw);
	}
```

```
        else // Avoid going the long-way round
        {
                Turret->Rotate(-DeltaRotator.Yaw);
        }
}

void UTankAimingComponent::Fire()
{
        if (FiringState == EFiringState::Locked || FiringState ==
EFiringState::Aiming)
        {
                // Spawn a projectile at the socket location on the barrel
                if (!ensure(Barrel)) { return; }
                if (!ensure(ProjectileBlueprint)) { return; }
                auto Projectile = GetWorld()->SpawnActor<AProjectile>(
                        ProjectileBlueprint,
                        Barrel->GetSocketLocation(FName("Projectile")),
                        Barrel->GetSocketRotation(FName("Projectile"))
                );

                Projectile->LaunchProjectile(LaunchSpeed);
                LastFireTime = FPlatformTime::Seconds();
                RoundsLeft--;
        }
}
```

```
#include "BattleTank.h"
#include "TankBarrel.h"


void UTankBarrel::Elevate(float RelativeSpeed)
{
        // Move the barrel the right amount this frame
        // Given a max elevation speed, and the frame time
        RelativeSpeed = FMath::Clamp<float>(RelativeSpeed, -1, +1);
        auto ElevationChange = RelativeSpeed * MaxDegreesPerSecond *
GetWorld()->DeltaTimeSeconds;
        auto RawNewElevation = RelativeRotation.Pitch + ElevationChange;
        auto Elevation = FMath::Clamp<float>(RawNewElevation,
MinElevationDegrees, MaxElevationDegrees);
        SetRelativeRotation(FRotator(Elevation, 0, 0));
}
```

```
#include "GridEditorCommands.h"

#define LOCTEXT_NAMESPACE "GridEditorCommands"

FName FGridEditorCommands::SquareModeName = "ToolMode_Square";

FName FGridEditorCommands::HexagonModeName = "ToolMode_Hexagon";

TMap<FName, TSharedPtr<FUICommandInfo> >
FGridEditorCommands::NameToCommandMap;

FGridEditorCommands::FGridEditorCommands()
```

```
        : TCommands<FGridEditorCommands>("GridEditor", NSLOCTEXT("Contexts",
"GridEditor", "Grid Editor"), NAME_None, FEditorStyle::GetStyleSetName())
{

}

FGridEditorCommands::~FGridEditorCommands()
{

}

void FGridEditorCommands::RegisterCommands()
{
        UI_COMMAND(SquareMode, "Mode - Square", "",
EUserInterfaceActionType::RadioButton, FInputChord());
        NameToCommandMap.Add(SquareModeName, SquareMode);
        UI_COMMAND(HexagonMode, "Mode - Hexagon", "",
EUserInterfaceActionType::RadioButton, FInputChord());
        NameToCommandMap.Add(HexagonModeName, HexagonMode);
}

#undef LOCTEXT_NAMESPACE

#include "Grid.h"
#include "GridManager.h"
#include "GridPainter/GridPainter.h"
#include "Components/DecalComponent.h"
#include "Kismet/KismetMathLibrary.h"
#include "Kismet/KismetSystemLibrary.h"

UGrid::UGrid()
{
        GridSize = 0.f;
        Height = 0.f;

        bVisible = false;

        GridInfo = nullptr;
        GridManager = nullptr;
        GridType = EGridType::Unknown;
}

UGrid::~UGrid()
{

}

FVector UGrid::GetCenter() const
{
        return FVector::ZeroVector;
}

void UGrid::SetGridSize(float Size)
{
        GridSize = Size;

        GridManager->GetGridPainter()->UpdateGridState(this);
}

float UGrid::GetGridSize() const
{
        return GridSize;
```

```
}

bool UGrid::Equal(const UGrid* R) const
{
	return Coord == R->Coord && FMath::Abs(Height - R->Height) <=
FLT_EPSILON;
}

void UGrid::SetVisibility(bool Visibility)
{
	if (bVisible != Visibility)
	{
		bVisible = Visibility;
		GridManager->GetGridPainter()->UpdateGridState(this);
	}
}

bool UGrid::GetVisibility() const
{
	return bVisible;
}

int UGrid::GetDistance(const UGrid* Dest) const
{
	return 0;
}

bool UGrid::IsEmpty() const
{
	return !GridInfo->HitResult.bBlockingHit;
}

FIntVector UGrid::GetCoord() const
{
	return Coord;
}

void UGrid::GetNeighbors_Implementation(TArray<UGrid*>& Grids)
{
	Grids.Reset();
}
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації і управління

УЗГОДЖЕНО
Керівник проєкту
_____ *Алла КОГАН*
(підпис) (вл. ім’я, прізвище)
“13” квітня 2020 р.

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)
“14” квітня 2020 р.

Система управління агентами у відео грі
виконавців

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 6110.01.000 ТЗ*

на 8 сторінках

Київ – 2020 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повне найменування системи: «Система управління агентами у відео грі».

Умовне позначення: «Система управління агентами».

Скорочена назва: «Система».

1.2 Найменування організації-замовника та організацій-учасників робіт

Замовник: кафедра автоматизованих систем обробки інформації та управління факультету інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Представник замовника: доцент кафедри АСОІУ Коган Алла Вікторівна.

Адреса замовника: м. Київ, п-кт Перемоги 37.

Виконавець: студент групи ІС-61з кафедри АСОІУ ФІОТ КПІ ім. Ігоря Сікорського Фокін Андрій Ігорович.

1.3 Перелік документів, на підставі яких створюється система

При розробці системи і створення проектно-експлуатаційної документації Виконавець повинен керуватися вимогами таких нормативних документів:

- ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;
- ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;

- ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплектність і позначення документів при створенні автоматизованих систем.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку виконання робіт: 15 лютого 2020 року.

Плановий термін закінчення виконання робіт: 31 травня 2020 року.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Призначення системи

Система призначена для управління агентами у відео грі за рахунок симуляцій штучного інтелекту кожного з агентів.

Систему можна використовувати як ігрову механіку і різних відео іграх або ж як основу для різних симуляцій, що включають переміщення певних об’єктів.

2.2 Цілі створення системи

Цілями створення є:

- спрощення процесу управління агентами;
- спрощення процесу створення відео ігор з використанням агентів.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТУ АВТОМАТИЗАЦІЇ

Об'єктом автоматизації є процес переміщення агентів.

Для того, щоб вирішити задачу переміщення агентів, необхідно пройти такі етапи:

- зчитати доступні площини переміщення агентів;
- визначити розміщення цілі актора;
- проаналізувати можливі шляхи переміщення до цілі і знайти оптимальний.

Для того, щоб автоматизувати цей процес, необхідно розв'язати задачу пошуку оптимального шляху.

По завершенню роботи ми отримаємо систему, яка дозволить автоматизувати процес управління агентами у відеогрі.

Розглянемо процес управління агентами після процесу автоматизації:

- визначити доступні площини переміщення агентів;
- визначити майбутні цілі агентів у цьому просторі;
- створити акторів;
- запустити процес симуляції.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Задача підготовки побудови системи управління агентами вимагає наявного простору переміщення та алгоритму знаходження цілі агентом .

Задача підготовки простору переміщення вимагає визначення можливих варіантів переміщення агента.

4.2 Вимоги до надійності

Система повинна адекватно реагувати на помилки застосування та видавати відповідні повідомленні користувачеві, а також записувати їх у log-файл.

Система повинна знаходити найоптимальніший шлях для всіх комбінацій положень акторів і їх цілей у заданому просторі. Після знаходження маршруту система повинна перемісти актора до його цілі.

4.3 Умови експлуатації

Для адекватної роботи системи необхідний пристрій з платформою, яка відповідає вимогам зазначеним в розділі 4.4 .

Усі користувачі системи повинні дотримуватися правил експлуатації електронної обчислювальної техніки.

4.4 Вимоги до складу і параметрів технічних засобів

Даний програмний продукт представлений у вигляді мобільного застосування і складається тільки з клієнтської частини.

Для коректної роботи гри на платформі Android потрібна версія операційної системи не нижче ніж 5.1.

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

У таблиці 5.1 наведено календарний план робіт та терміни їх виконання.

Таблиця 5.1 – Календарний план виконання робіт

№ з/п	Назва етапів створення продукту	Строк виконання
1.	Вивчення рекомендованої літератури	15.02.2020 р.
2.	Аналіз існуючих методів розв’язання задачі	22.02.2020 р.
3.	Постановка та формалізація задачі	01.03.2020 р.
4.	Розробка інформаційного забезпечення	15.03.2020 р.
5.	Алгоритмізація задачі	22.03.2020 р.
6.	Обґрунтування використовуваних технічних засобів	30.03.2020 р.
7.	Розробка програмного забезпечення	20.04.2020 р.
8.	Налагодження програми	27.04.2020 р.
9.	Виконання графічних документів	04.05.2020 р.
10.	Оформлення пояснювальної записки	18.05.2020 р.
11.	Подання ДП на попередній захист	22.05.2020 р.
12.	Подання ДП на основний захист	01.05.2020 р.
13.	Подання ДП рецензенту	05.06.2020 р.

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ СИСТЕМИ

6.1 Види випробувань

Види випробувань узгоджуються із замовником до проведення випробувань. Здача - прийом робіт виконується поетапно на комп'ютерах замовника в аудиторіях кафедри АСОІУ у відповідності з робочою програмою та календарним планом.

Усі програмні продукти, що створюються в рамках даної системи передаються замовнику як у вигляді готових модулів, так і у вигляді вихідних кодів, представлених в електронній формі.

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

ID перевірки:
1004063837

Дата перевірки:
16.06.2020 04:18:31 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
16.06.2020 04:51:53 EEST

ID користувача:
77149

Назва документу: **Fokin_isz61_3**

ID файлу: **1004076805** Кількість сторінок: **43** Кількість слів: **6077** Кількість символів: **42675** Розмір файлу: **2.56 MB**

6.58% Схожість

Найбільша схожість: **4.21%** з джерело бібліотеки. ID файлу: **12191126**

2.52% Схожість з Інтернет джерелами	28	Page 45
6.58% Текстові збіги по Бібліотеці акаунту	154	Page 45

0.43% Цитат

Цитати	1	Page 46

Вилучення переліку посилань вимкнено

0% Вилучень

Вилучений текст відсутній

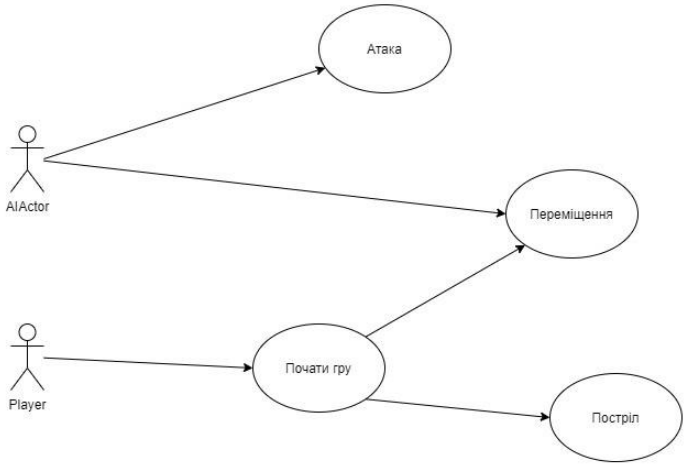
Підміна символів

Не знайдено замінених символів

Графічний матеріал
до дипломного проєкту

на тему: Система управління агентами у відео грі

Київ – 2020 року



					ДП 6110.03.000 ССВ	
Зм.	Арк.	№ документа	Підпис	Дата		Літера / Маса / Масштаб
Розробив		Фокін А.І			Схема структурна варіантів використання	
Перевірив		Коган А.В.				Аркуш 1 / Аркушів 1
Т. кон.						
Н. кон.		Тєлишева Т.О.			Система управління агентами у відео грі	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з61
Затвердив		Коган А.В.				

ДП 6110.08.000 ССК

					ДП 6110.08.000 ССК
Зм.	Арк.	№ документа	Підпис	Дата	Літера / Маса / Масштаб
Розробив		Фокін А.І			Схема структурна класів програмного забезпечення
Перевірив		Коган А.В.			Аркуш 1 / Аркушів 1
Т. кон.					
Н. кон.		Тєлишева Т.О.			Система управління агентами у відео грі
Затвердив		Коган А.В.			КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з61

ДП 6110.07.000 ССП

					ДП 6110.07.000 ССП
Зм.	Арк.	№ документа	Підпис	Дата	Літера / Маса / Масштаб
Розробив		Фокін А.І			Схема структурна послідовності
Перевірив		Коган А.В.			Аркуш 1 / Аркушів 1
Т. кон.					
Н. кон.		Тєлишева Т.О.			Система управління агентами у відео грі
Затвердив		Коган А.В.			КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з61

ДП 6110.06.000 СК

					ДП 6110.06.000 СК
Зм.	Арк.	№ документа	Підпис	Дата	
Розробив		Фокін А.І			Схема компонентів
Перевірив		Коган А.В.			
Т. кон.					
Н. кон.		Тєлишева Т.О.			Система управління агентами у відео грі
Затвердив		Коган А.В.			КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з61

Літера	Маса	Масштаб
Аркуш 1	Аркушів 1	

ДП 6110.02.000 СО

					ДП 6110.02.000 СО	
						Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема організаційної структури	
Розробив		Фокін А.І				Аркуш 1 / Аркушів 1
Перевірив		Коган А.В.				
Т. кон.						
Н. кон.		Тєлишева Т.О.			Система управління агентами у відео грі	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з61
Затвердив		Коган А.В.				

ДП 6110.04.000 КЕ

					ДП 6110.04.000 КЕ	
						Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	
Розробив		Фокін А.І				Аркуш 1 / Аркушів 1
Перевірив		Коган А.В.				
Т. кон.						
Н. кон.		Тєлишева Т.О.			Система управління агентами у відео грі	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з61
Затвердив		Коган А.В.				